

**NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series**

NASA-IVV-96-003
WVU-SRL-96-003
WVU-SCS-TR-96-12
CERC-TR-RN-96-007

*11-1-96
038 160*

**Experience Report: The Use of Functional Flows to Provide
an Alternate Perspective for IV&V**

by Edward A. Addy and Lynn J. Simms




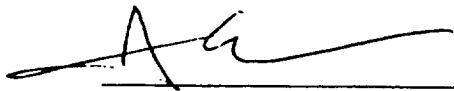
National Aeronautics and Space Administration



West Virginia University

According to the terms of Cooperative Agreement #NCCW-0040,
the following approval is granted for distribution of this technical
report outside the NASA/WVU Software Research Laboratory


George J. Sabolish Date
Manager, Software Engineering


John R. Callahan Date
WVU Principal Investigator

Experience Report: The Use of Functional Flows in IV&V

Edward A. Addy

NASA/WVU Software Research Laboratory
100 University Drive
Fairmont, WV 26554
eaddy@wvu.edu

Lynn J. Simms

Logicon RDA
P.O. Box 1420
Dahlgren, VA 22448
lsimms@logicon.com

ABSTRACT

One of the characteristics of Independent Verification and Validation (IV&V) that help make it an effective systems engineering technique is the fresh perspective it brings to the software development effort. This fresh perspective can be enhanced if the IV&V tasks are organized to cut across the work breakdown structure of the software development tasks. This paper describes an IV&V technique that uses functional flow descriptions as a basis for IV&V analysis. The functional flows generally pass through multiple software modules, where the module is the basic unit of responsibility for the development team. The use of functional flows as a basis for IV&V analysis allows the IV&V team to have a different perspective from that of the development team. This method can also be used to determine the critical areas of the software and to focus the IV&V effort toward these critical areas. The method further provides a framework for interaction for the IV&V analysts who have responsibilities in different areas of the software.

KEY WORDS

functional flow, perspective, IV&V, nuclear safety, software accreditation, software engineering, software safety, validation, verification

INTRODUCTION

The objective of an IV&V effort is to develop an independent assessment of the software quality and to determine if the software satisfies critical systems requirements¹⁰. In an effort to meet this objective,

IV&V projects often seek to analyze the software using methods, tools, and processes that are complementary to but distinct from those used by the developer. The complementary viewpoints utilized by IV&V include efforts to determine conditions that might not have been tested by the developing organization, including boundary constraints, off-nominal conditions, error paths, and user scenarios. These viewpoints enable the IV&V team to have a different perspective of the software from that of the development team.

One method to provide a different perspective is to organize the analysis effort so that an IV&V analysis area involves software that cuts across the basic units constructed by the developer. Since IV&V focuses on critical system requirements, the cuts focus on the software that performs functions that are critical to the operation of the system. This experience report describes how functional flows are used to provide a structure for IV&V analysis. It describes the methods used and the additional advantages gained by the IV&V team through the use of functional flows.

ISNSA ENVIRONMENT

This work is performed as part of the Independent Software Nuclear Safety Analysis (ISNSA) of the Tomahawk Weapon System (TWS). The TWS and other weapon systems that are capable of using nuclear weapons are developed in accordance with the United States Department of Defense (DoD) standards for nuclear safety, Safety Studies and Reviews of Nuclear Weapons, DoD Directive 3150.2⁴. The Navy implements the DoD nuclear safety requirements within the Tomahawk system through the Tomahawk Nuclear Safety and Certification Program, PDA-14INST 8020.1A⁷.

The work described in this paper was performed while both authors were employed by Logicon, Inc. The work was performed under contract number N00019-90-C-0050. Support for writing the paper was provided by NASA under contract number NCCW-0040.

ISNSA is part of this program, and is performed to ensure the software does not cause or contribute to a violation of nuclear safety standards.

The focus of ISNSA is nuclear safety, so the analysis is not focused on general safety or on performance issues. However, ISNSA uses many of the same techniques and methods used by the software safety community and by general IV&V⁹. In addition, ISNSA is the only activity within the TWS development process that provides an independent assessment of the software. ISNSA finds and documents many problems related to general safety and to performance as a collateral result of analyzing the software for nuclear safety.

The Tomahawk Nuclear Safety and Certification Program identifies a set of system actions that must be controlled properly in order to ensure the weapon system is in conformance with DoD nuclear safety standards⁷. These critical actions are implemented and controlled within the system through some combination of hardware, software, or administrative components. The portions of the system that implement and control the critical actions are termed "critical factors."

The ISNSA team has the responsibility to identify the software that is a part of any critical factor. The software functions that are included in critical factors are termed "critical functions", and this critical software receives priority for analysis. A critical factor may be implemented across multiple software modules, tasks, and computers.

The system considered by the ISNSA team consists of three major components: the Weapon Control System (WCS), the Vertical Launch System (VLS), and the missile. This report deals with ISNSA performed on the TWS system and on the WCS subsystem for several major versions. The overall TWS system ISNSA deals with a high-level view of the system, focusing on external interfaces and the interfaces between the various hardware and software components of the weapon system. The WCS subsystem ISNSA goes to a more detailed level, extending to the code implementation.

The WCS subsystem software is developed in conformance to the Defense System Software Development standard, DOD-STD-2167A³, and consists of FORTRAN and Assembly code in a real-time, interrupt-driven system. The software has a basic hierarchical structure, with multiple tasks. Each task issues a sequence of calls to subroutines and

functions, and executes on a separate thread. The tasks communicate through messages and shared memory locations, and have priority assignments that determine scheduling for execution. The individual tasks in the code often have a correspondence to particular sections within the program specification document.

The ISNSA team analyzing the WCS subsystem is organized according to the structure of the software. Each analyst is assigned particular sections of the program specification, and is expected to follow that area of the software through requirements, design, implementation, and test phases of development. The developing organization is also organized along this same basic hierarchical structure.

FUNCTIONAL FLOWS

Functional flows are a means of expressing software requirements, design or implementation in a diagram form. A function flow is a block diagram that depicts the relationships between the system components that are required to accomplish a particular function, including inputs, outputs, constraints, and control flow.⁶ The diagrams are used by analysts to search for ambiguities, inconsistencies, incompleteness, and areas of potential weakness.

The ISNSA approach combines functional flow block diagrams with the Tomahawk nuclear safety critical factors. The ISNSA team prepares functional flow diagrams that depict the flow of data and control in the software necessary to perform each of the critical actions⁵. The resulting functional flows are termed Critical Factor Flow Diagrams (CFFDs). The CFFDs are designed to identify and analyze the system elements and interdependencies of system elements that are necessary to the realization of the critical factor being diagrammed.

CFFDs are created at two levels: the TWS system level and the WCS subsystem level. These two types of CFFDs are constructed in formats created by the ISNSA team. TWS system CFFDs are constructed in a format most suitable for representing the relationships between subsystem-level software and hardware components. WCS subsystem CFFDs are constructed in a format most suitable for representing the relationships between WCS software components and between WCS software components and software and hardware components in other subsystems. The WCS appears as one subsystem in the system CFFDs.

Notes within the WCS area of the system CFFDs indicate corresponding sections of WCS subsystem CFFDs. The critical factor being represented is used in the naming convention for both TWS and WCS CFFDs, so flows related to the same critical factor can also be identified in this manner. However, the portions of the WCS subsystem CFFDs that do not involve elements from other subsystems do not appear in system CFFDs. Also, while all critical factors must be addressed at the system level, not all critical factors involve the WCS subsystem.

Special symbol formats used in the system diagrams include rectangles for digital messages across system component interfaces and squares for hardware devices and monitors. The subsystem CFFDs use rectangles to represent interfaces to other subsystems within the weapon system. Symbols used by both the system and subsystem CFFDs include AND gates and OR gates, and diamonds for decision boxes that alter the path to be taken for different data values. Lines in both diagrams represent control flow, message flow, or signal flow as appropriate. Figure 1 shows a sample System CFFD, and Figures 2a-2d contain a sample Subsystem CFFD, all taken from the ISNSA work. Figure 3 indicates the meanings of the various symbols used in the functional flows.

A brief description of the sample CFFDs is given below, in order to provide a general understanding of the flows being diagrammed. Many details, such as the message information and values of the variables, are not described since this paper addresses the concept of the CFFDs rather than the particulars of the TWS implementation.

The system level functional flows created by the ISNSA team are generated from requirements and interface control documents, while the subsystem level CFFDs are based on source code implementation. The symbols and formats are chosen as appropriate for the level of detail and the relationships within the respective functional flows. These examples should not preclude the use of functional flows at other levels of system development, nor should they prescribe the use of these symbols and formats for all functional flows.

System CFFDs

TWS system CFFDs depict the functional processing of the critical factors by showing actions and

interactions of the various subsystems. The message flows, device control and device status are shown in these diagrams, as well as the Boolean combination of events necessary to cause actions. The system CFFDs visually divide the flow of messages and status data across distinct sections of the diagram. Each section represents a subsystem or hardware component, including WCS subsystem components, launch system components and missile components.

Figure 1 depicts the flow for the processing to request missile status information. The message originates in the WCS subsystem, and is passed through two subsystems of the VLS to reach the missile. The flow of return information and the processing involved is also shown in the diagram.

The information in the WCS subsystem section of the system CFFD shows the particular subsystem CFFD number that sends or receives the data from that system CFFD. Critical timing requirements are noted where flow of information has to occur within a particular window of time.

Subsystem CFFDs

The WCS subsystem CFFDs depict the flow of data and control through the hierarchical structure of the software. Each WCS CFFD is broken into layers of subflows, where the lowest level subflow represents the path and actions taken during one pass of a task. Each subflow of a CFFD culminates in a variable(s) being set or a message being transmitted that is a step toward the accomplishment of the critical action associated with the critical factor. Variables being set to the values indicated by the subflow appear as input conditions in other subflows, in the same or other CFFDs. Each page of the subflow shows the routines, calling structure, and variables used within a single task.

The Boolean events leading into the AND gate of a routine identify a specific path in the routine that is executed in this critical factor. This path may also appear in other pages of a CFFD or in other CFFDs. Other paths in a routine often occur in the same or other CFFDs.

Figures 2a-2d show the CFFD for the processing to grant permission to fire for a plan, which is a specific step in the launch sequence. The Permission To Fire CFFD has only one layer of subflows. Granting

permission to fire involves the four basic steps listed below.

1. determining that the plan has advanced to the proper state to place an operator prompt on a queue
2. displaying the prompt
3. recognizing that the operator has responded properly to the prompt
4. setting the state of the plan to indicate that permission has been granted.

This processing requires action in four major task threads, which results in the CFFD having four pages. Figure 2a shows the processing within Task ELSST for the Request Permission to Fire Prompt to be placed on the prompt queue. Figure 2b shows the task which acts upon the operator request to display the prompt. Figure 2c shows the display processing within the third task when the operator turns the Permission to Fire Key. Figure 2d shows the final processing within a subsequent pass of Task ELSST to set the state of the subsystem to indicate that permission has been granted for the plan.

The Boolean combination of the variables evaluated in the critical path are shown on the flow, as well as the values to which critical variables are set. Each page results in the setting of a critical variable or the output of a message, and this result is used to name the individual pages of the subflow. The code variable name, the value assigned or being evaluated, and a description of the condition holding when the variable assumes that value are all listed on the subflow for both assignments and evaluations. Great pains are taken to ensure consistency in the descriptions across all the CFFDs, so that identical conditions are described using the same phrasing. This allows text searches to be used to locate areas of interest.

CONSTRUCTION PROCESS AND USE

Some CFFDs exist for the previous version of the system and the subsystem. The system diagrams and some subsystem diagrams are assigned to analysts already experienced in those portions of the ISNSA. These analysts are able to use the previous diagrams as baselines and update them for the differences that occurred. These changes are known to these analysts from the results of difference programs run on the subsystem code, or from differences in the Performance Specifications or Interface documents of

system components. The efforts of these analysts are focused on updates first, and then on studying the flows for any subtle impacts the changes might have caused. The CFFDs become meaningful, visual tools for determining the impact of change.

The ISNSA group also has several new analysts. These analysts are not able to use the existing CFFDs as baselines because they do not yet know the code deeply enough to detect subtle problems that changes might cause. The new analysts instead start constructing the CFFDs from the beginning. These analysts study the assigned code, create a basic CFFD, and then compare their diagram to the baseline CFFDs and known difference listings as the experienced analysts do. Creating completely new CFFDs when old CFFDs already exist may seem to be duplicative work. However, the analysts find that this one-time investment serves them well in better understanding the flows of the program and in helping them to detect the more subtle errors. The actual time spent in this integrated process is comparable to the sum of a "pure code study phase" plus a "pure update CFFD" phase.

The TWS CFFDs are created from information in interface control documents and from operational flow diagrams in the high-level system specifications. The WCS CFFDs are based on code analysis and the use of a flow chart generator. Each Boolean combination of variables feeding into a module AND gate represents a particular path through the module(s). The represented path terminates in the call to the next module or in the statements that result in the output indicated by the CFFD.

The analysts use the CFFDs not only to verify the basic processing of each critical factor, but also to investigate error processing and off-nominal cases. By examining each variable involved in the CFFD, the analyst can consider the possibility of conditions that could affect the value of that variable. The CFFDs also present a visual representation of the processing by which the analyst can consider the completeness of each flow, and if all cases within each situation are handled by the software.

The CFFDs assist in communicating about the processing of the critical factors to other ISNSA analysts and to ISNSA management. The CFFDs are used in determining the impact of mistakes in one area of the software upon other critical processing. The CFFDs also serve as the basis for formal inspections by the ISNSA team.

In addition to aiding the analysis of each area by the analyst assigned responsibility to that area, construction of the CFFDs provides a framework for interaction. The analysts have to cooperate with each other to develop a set of CFFDs that are complete and consistent. Many of the CFFDs involve multiple tasks that may be assigned to different analysts, and these analysts have to work together to construct the complete CFFD. This cooperation provides a basis for the analysts to learn about program areas outside their assigned responsibility.

COMPARISON WITH OTHER METHODS

Functional flows have aspects that are common with other analysis methods used in software safety, including fault trees, event trees, and failure mode and effects analysis. Functional flows also share some commonalities with testing approaches such as operational profiles and user scenarios. This section discusses the similarities, differences, and relationships between functional flows and these other methods.

Fault trees are frequently used for system safety analysis, and show the combination of events that could cause or contribute to the occurrence of an undesirable state. Fault trees are depicted using a tree structure with AND and OR nodes, and usually contain hardware conditions, software conditions and human interactions. A fault tree is the logical representation of the relationship of primary events that lead to a specified undesirable event. The construction of the fault tree is top-down, in that the undesirable event is the root of the tree and the logical combination of subevents are used to map out the tree until the basic initiating events are reached.

Event trees appear very similar to fault trees, and may use the same representations. However, event trees are used to identify the effects of an event instead of the causes. Rather than starting from a particular system event and working backwards to the causes, an event tree traces a primary event forward in order to determine the consequences of the event. Event tree analysis is inductive as opposed to the deductive fault tree analysis⁸.

Failure Mode Effects Analysis (FMEA) is an inductive method used to systematically consider the effects of all failure modes. The system is decomposed into its component parts, each of which has known or anticipated failure modes. Each failure

mode is analyzed, with an examination of the cause, effect, severity, probability and prevention or mitigation of each failure mode.

Fault tree analysis, event tree analysis and FMEA are all methods of software safety analysis, as is functional flow analysis. However, due to its focus on functional processing rather than strictly on the cause or effect of a fault, functional flows can be extended beyond software safety into the more general area of software assurance. The derivation of the CFFDs tends to include both top-down and bottom-up techniques; the critical system action accomplished by the critical factor is the top of the functional flow, but the functional flow is generated by examining the processing of the various software components. While event trees have a tendency to expand rapidly as combinations of events cause different effects, the critical factor serves to constrain the scope of the functional flow.

Fault trees and event trees are both used to show system states or conditions that are the cause or effect of a particular system state. The emphasis of fault trees and event trees is on determining the states that can occur. Functional flows emphasize the process of achieving various states of the system, showing alternate execution paths and control flows that result in the same state. This additional information is useful if the analysis is being conducted to provide assurance that the software is performing correctly and not just to determine and avoid fault conditions.

Functional flows are not directly related to FMEA tables, but could prove useful in the analysis of failure modes. Functional flows provide insights into relationships between software components that could assist in determining the causes and effects of a given fault mode. Since the functional flows are constructed for critical processing, this would also assist in determining the severity of a particular fault.

BENEFITS

The ISNSA effort gains several benefits from the use of functional flows as a basis for analysis. These benefits are both technical and organizational in nature, and are described in the paragraphs below.

1) Perspective

A stated objective in using the CFFDs as a basis for IV&V analysis is to provide the analysts a

perspective that differs from the perspective of the developer. The perspective provided by the CFFDs allows the ISNSA analysts to consider situations and abnormal conditions that are not easily seen during the construction of the software. An example of this is the discovery of a path in which a single variable was responsible for indicating that a critical factor had been completed, thus allowing the next critical factor to begin. This violated the nuclear safety constraint that no single fault be allowed to initiate a critical factor.

The perspective provided by the CFFDs is not simply different from that of the developer, but is useful to the IV&V analysts in understanding the software. The CFFDs give the analysts insight into the internal functioning of a module, into the overall flow, and in particular to interfaces between modules. In addition, this insight into the flow of the program is from an operator's standpoint (closer to the critical factors) rather than from a pure hierarchical (code study) or requirements based (specification study) view.

2) High-Risk Focus

ISNSA attempts to focus its efforts on that portion of the software that deals directly with the control of nuclear weapons. (The ISNSA team also recognizes that seemingly unrelated sections of the software can have an impact on nuclear safety, and that some minimal examination is necessary for all software in the system¹.) The CFFDs provide a natural structure for giving priority to the software that is involved with the critical factors and is therefore most directly related to nuclear safety.

Although most systems do not have a predetermined set of "critical factors," many systems do have a set of actions that are critical to the proper operation of the system. The system components that implement and control those actions serve as the critical factors for that system, and the software components involved in those critical factors are the critical functions for that system. This concept can be applied not only to safety applications outside of nuclear weapon control systems, but to non-safety applications as well.²

3) Test Cases

Each functional flow depicts one or more operational scenarios for the system. These scenarios serve as a basis for writing test cases dealing with critical processing. Since many of the safety requirements also deal with specific critical factors, the test cases are appropriate for ISNSA. Having the flows in a graphical format also allows the analysts to perform

pen and paper simulations of the processing using a range of inputs. The ability to consider a range of inputs is also useful in determining the exact values to be used as inputs for testing, so that extremes and problematic inputs can be used.

4) Cross-Training

The primary assignment given to each analyst generally leads to the analyst being responsible for a software module that consists of one or more tasks and the calling structure beneath the tasks. The WCS subsystem CFFDs provide a forum for the analysts to consider and discuss interactions between the modules. This collaboration allows each analyst to learn about the structure and function of software outside their primary area of responsibility, and to so gain a better understanding of the program as a whole. This whole-program view, linked to a manageable amount of analyst responsibility, allows analysts to envision problems that could not be seen when considering only a portion of the software. One example of this was a variable set in one section of the software after verification that the proper conditions were satisfied, but then changed in another section without reverification to ensure that the new setting met all required conditions.

CONCLUSION

The use of CFFDs proved valuable as one of the methods used during ISNSA. By following a thread through multiple modules, this technique allowed the analysts to consider situations and combinations of events that would not normally be considered during the development of the software. This method was one of several used in ISNSA to focus the analysis on areas of high-criticality. In addition, the CFFDs provided a structure by which the analysts could learn about and discuss interactions and interfaces with software outside their immediate area of responsibility.

Although this method was used specifically within ISNSA, functional flows could be applied to both other areas of software safety analysis and to software assurance in general. The functional flows provide a framework for assurance activities to focus on the areas of software that perform activities important to the system user.

ACRONYMS

CFFD	Critical Factor Flow Diagram
DoD	Department of Defense
FMEA	Failure Mode Effects Analysis
ISNSA	Independent Software Nuclear Safety Analysis
IV&V	Independent Verification and Validation
TWS	Tomahawk Weapon System
VLS	Vertical Launch System
WCS	Weapon Control System

REFERENCES

1. Addy, Edward, "A Case Study in Isolation of Safety Critical Software", *Proceedings of the Conference on Computer Assurance 1991*, ACM Press, June 1991.
2. Addy, Edward, "Methodology of Independent Software Nuclear Safety Analysis", *Proceedings of the Fifth International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, November 1994.
3. DoD, *Defense System Software Development*, DOD-STD-2167A.
4. DoD, *Safety Studies and Reviews of Nuclear Weapons*, DoD Directive 3150.2, 1984.
5. Logicon, Inc. for Cruise Missiles Project and Unmanned Aerial Vehicles Joint Project, Washington, DC, *Sea Launched Cruise Missile, TOMAHAWK Weapon System MK 37, Independent Nuclear Safety Analysis, Critical Function Flow Diagrams, for Post Block III*, LSIS940012-CMV127, December 1994.
6. Makowsky, Lawrence C., A Guide to Independent Verification and Validation of Computer Software, USA-BRDEC-TR//2516, US Army Belvoir RD&E Center, June 1992.
7. Naval Air Systems Command, Tomahawk Nuclear Safety and Certification Program, PDA-14INST 8020.1A, 1989.
8. Place, Patrick R. H., and Kang, Kyo C., *Safety-Critical Software: Status Report and Annotated Bibliography*, Technical Report CMU/SEI-92-TR-5, ECS-TR-93-182, Software Engineering Institute, Carnegie Mellon University, June 1993.
9. Wallace, Delores R., and Fujii, Roger, *Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards*, NIST Special Publication 500-165, National Institute of Standards and Technology, September 1989.
10. Wallace, Delores R., and Fujii, Roger U., "Software Verification and Validation: An Overview," *IEEE Software*, May 1989, pp. 10- 17.

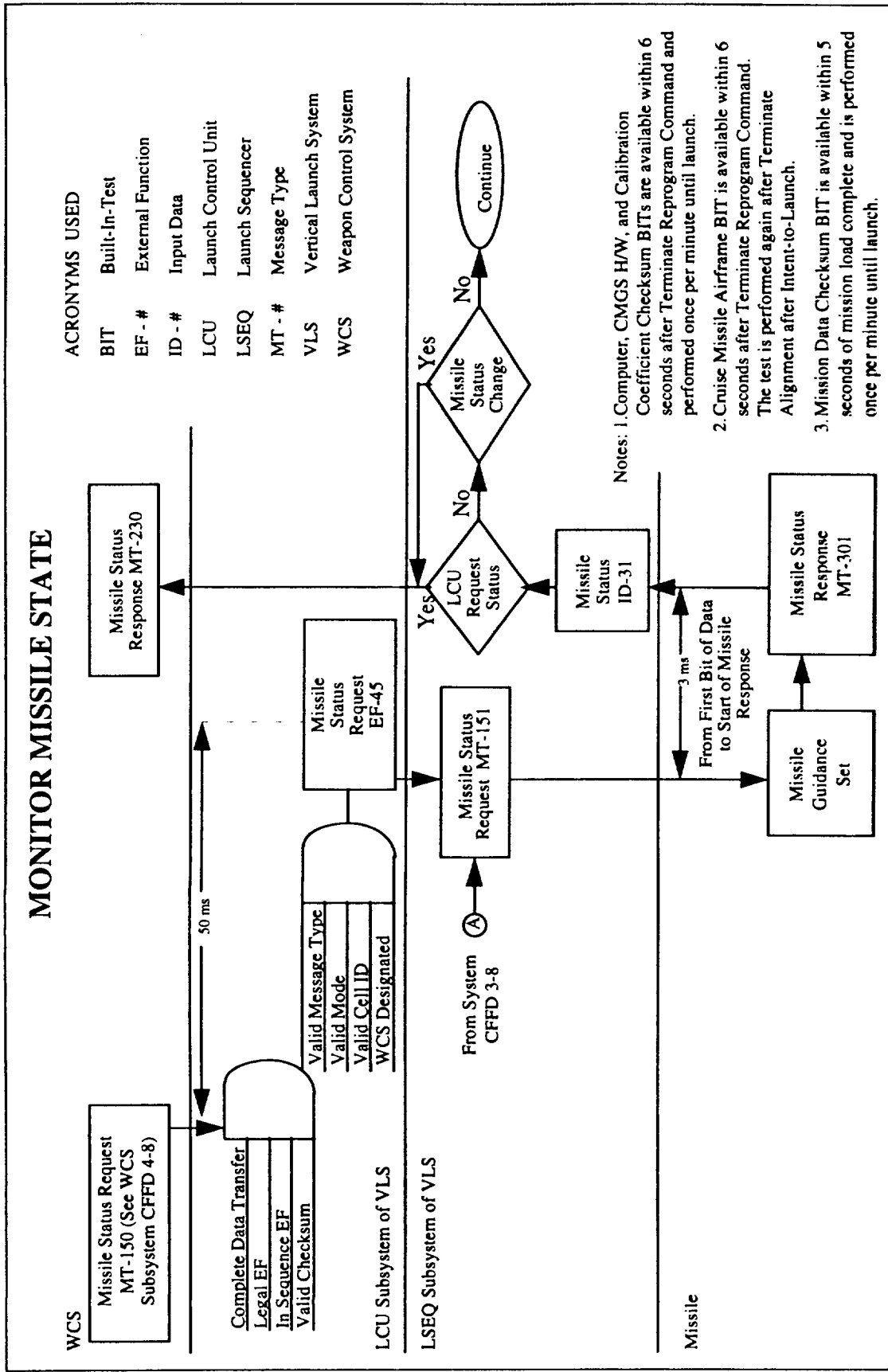


Figure 1 - SAMPLE SYSTEM CFFD: SYSTEM CFFD 3-9, Page 1 of 1

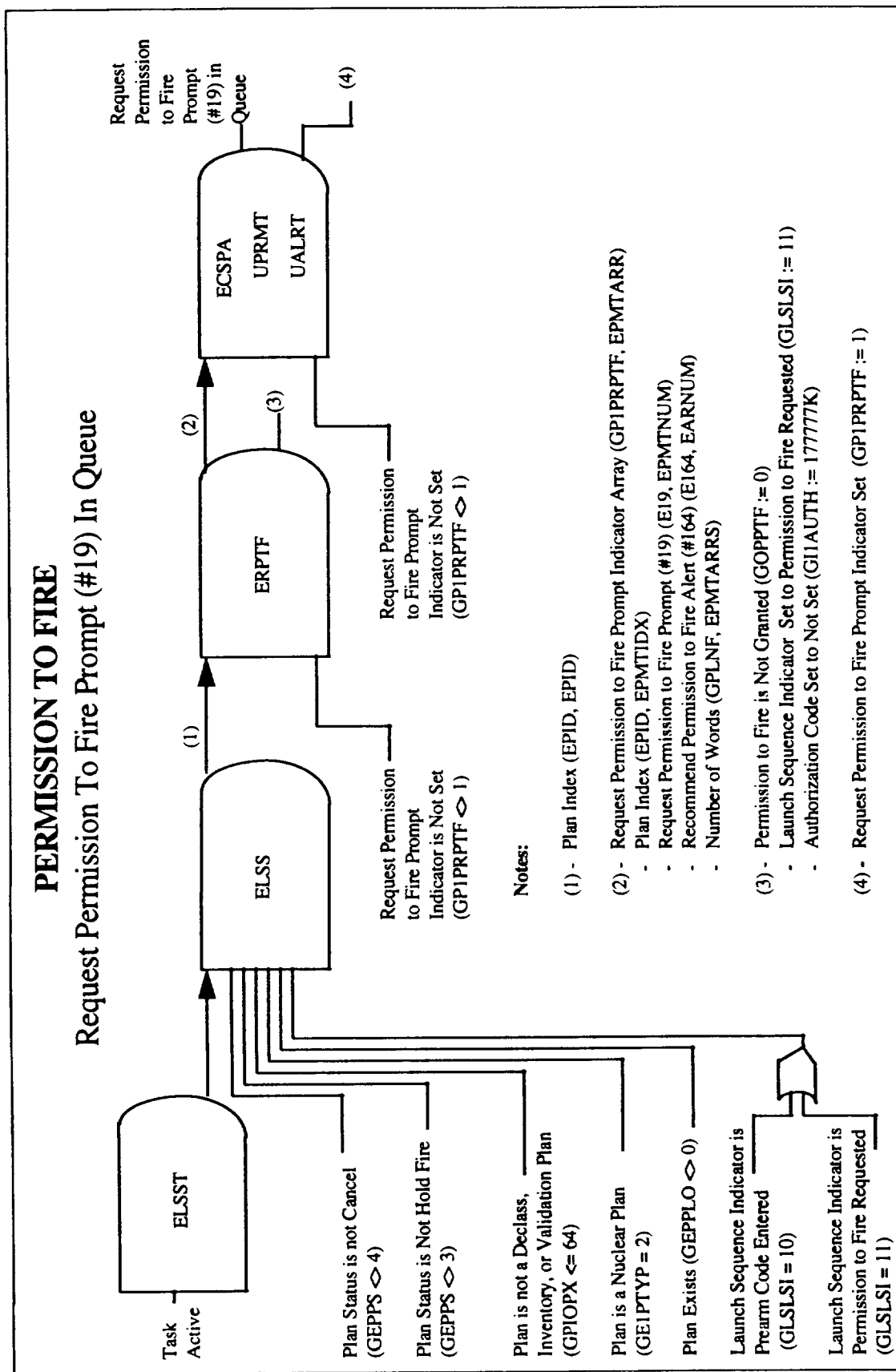


Figure 2a - SAMPLE SUBSYSTEM CFFD: WCS CFFD 4-7, PAGE 1 of 4

PERMISSION TO FIRE

Request Permission To Fire Prompt (#19) Displayed

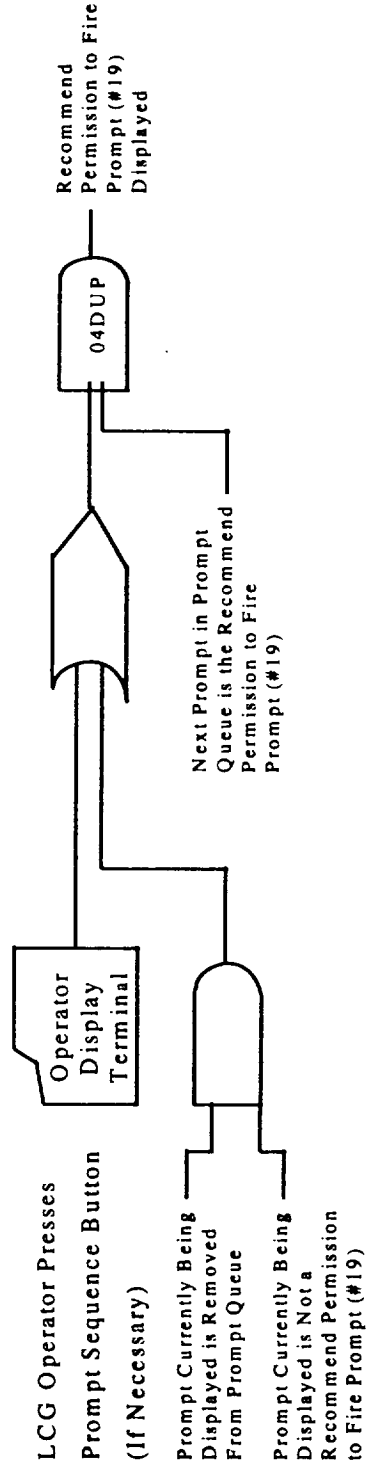


Figure 2b - SAMPLE SUBSYSTEM CFFD: WCS CFFD 4-7, PAGE 2 of 4

PERMISSION TO FIRE

Operator Turns Permission To Fire Key Permission to Fire Granted

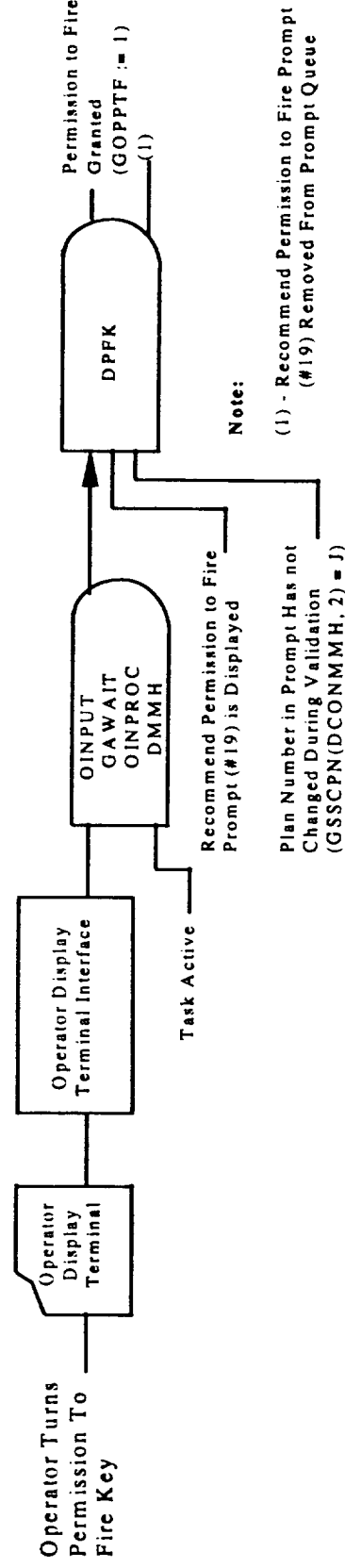


Figure 2c - SAMPLE SUBSYSTEM CFFD: WCS CFFD 4-7, PAGE 3 of 4

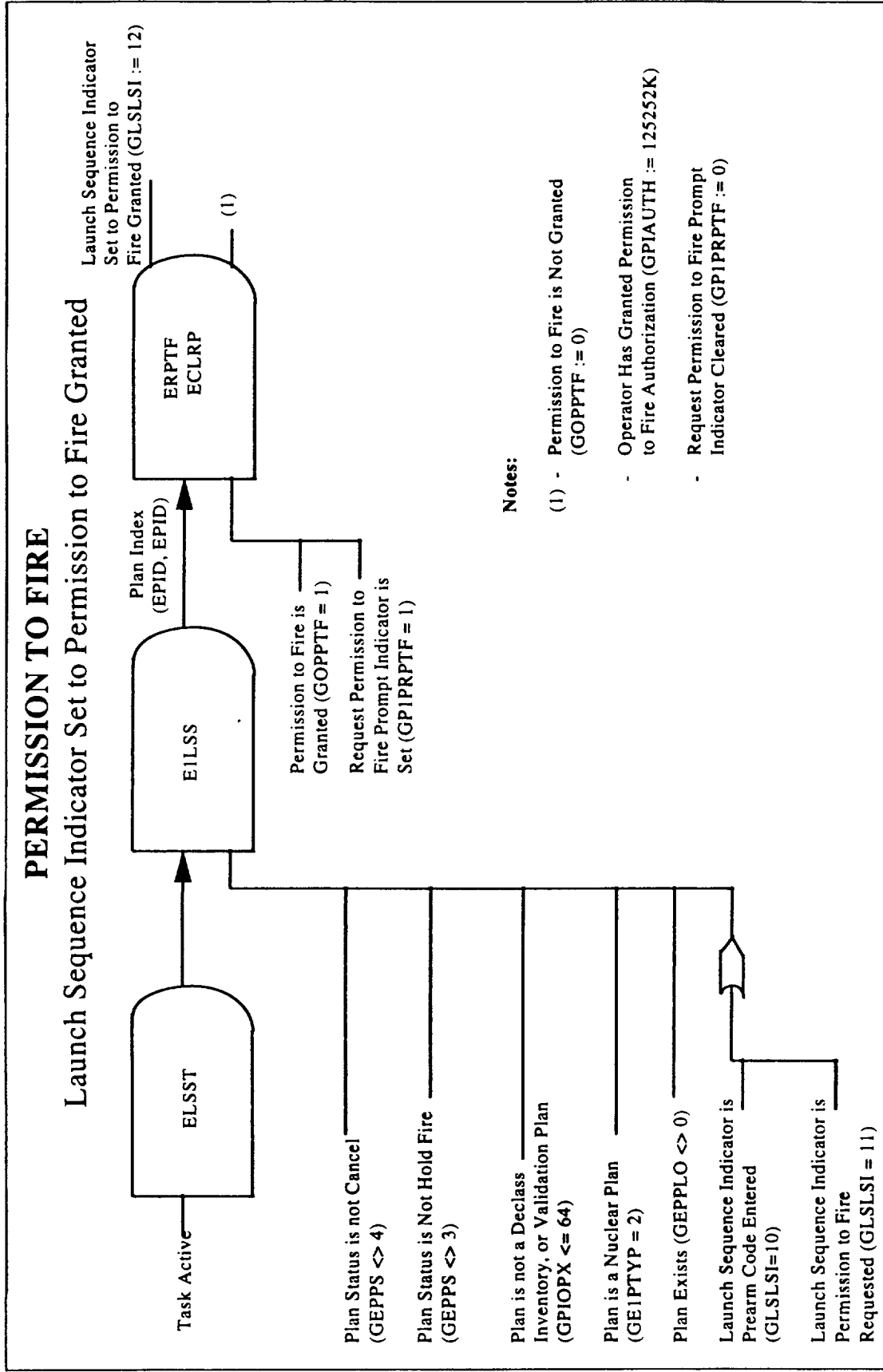














Figure 2d - SAMPLE SUBSYSTEM CFFD: WCS CFFD 4-7, PAGE 4 of 4

SYMBOLS USED IN THE FUNCTION FLOW DIAGRAMS

COMMON	SYSTEM	SUBSYSTEM
AND Gate ¹ 	Information Flow 	Control Flow ² 
OR Gate 	Timing Constraints 	Information Flow ³ 
Continuation 	Hardware Device or Monitor 	Operator Interface 
Decision Gate 	Message 	Subsystem Interface 
= Boolean Equality	◇, <, >, <=, >= Boolean Inequalities	:= Assignment

¹ The AND gate in the subsystem CFFDs contains the modules performing the checks and assignments.

² Formal and actual arguments are listed for subroutine calls.

³ Data conditions and assignments are described by a phrase followed by a Boolean condition using constants and actual variable names.

Figure 3 - CFFD SYMBOLS